# An Infrastructure for Passive Network Monitoring of Application Data Streams

Deb Agarwal, José María González, Goujun Jin, Brian Tierney

*Computing Sciences Directorate*
*Lawrence Berkeley National Laboratory*
*University of California, Berkeley, CA, 94720*

**Abstract**

*When diagnosing network problems, it is often desirable to have a view of traffic inside the network. In this paper we describe an infrastructure for passive monitoring that can be used to determine which segments of the network are the source of problems for an application data stream. The monitoring hosts are relatively low-cost, off-the-shelf PCs. A unique feature of the infrastructure is secure activation of monitoring hosts in the core of the network without direct network administrator intervention.*

## 1.0 Introduction

Achieving the goals of high performance distributed data access and computing requires extracting the best possible performance from the networks. This requires not only end-to-end network performance information, but also data from the interior of the network. Without this data, the user or network engineer is often unable to diagnose problems encountered in the end-to-end network path.

The passive monitoring system described in this paper, called Self-Configuring Network Monitor (SCNM), uses special activation packets to collect this data. These activation packets automatically activate monitors deployed at the layer three ingress and egress routers of the wide-area network, and at critical points within the site networks. Monitoring output data is sent back to the application data source or destination host. No modifications are required to the application or network routing infrastructure in order to activate monitoring of traffic for an application. This ensures that the monitoring operation does not add a burden to the network's administrator.

The hardware infrastructure for SCNM is designed to be easy to install and administer securely. Figure 1 shows a typical configuration between two application hosts, or *end hosts*, across a WAN. A read-only tap is placed on the DMZ between the site border router and the ISP router, and the monitoring host, which we call the SCNM *monitoring host*, is connected to this tap. The SCNM monitoring host has an additional network interface

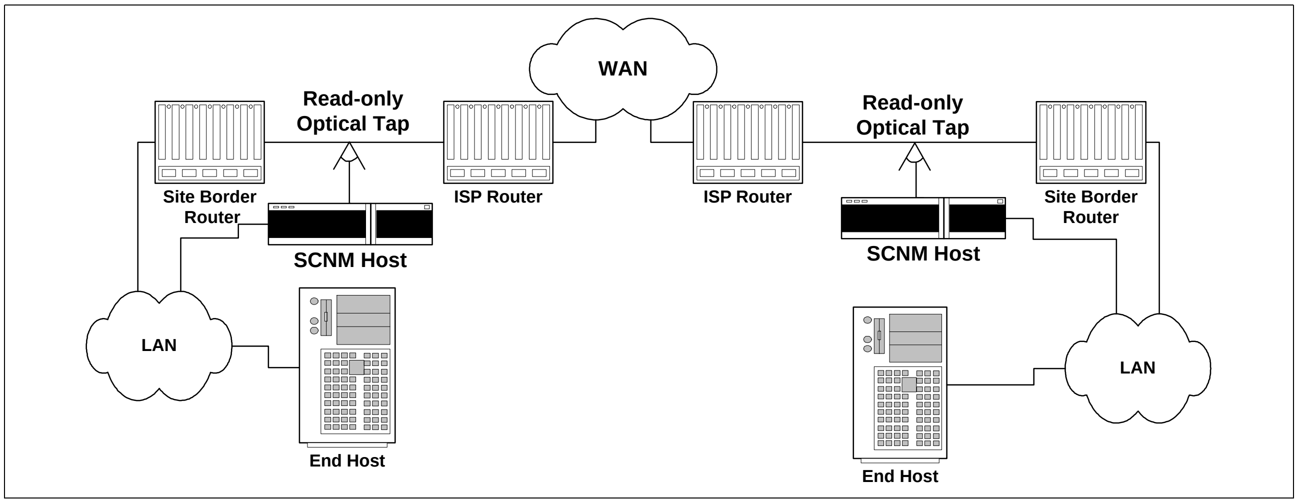

Figure 1: Typical Installation

(usually on an internal network) used for administering the SCNM host and sending monitoring output data.

## 2.0 Related Work

There are several passive network monitoring tools that are currently available. The Simple Network Management Protocol (SNMP) provides an interface to traffic statistics, usually gathered by passive sensors from routers and end hosts. A drawback of SNMP is that the requestor must have a community string or router access to get at the data. Many networks use the Multi Router Traffic Grapher (MRTG) [13] to collect and publish the SNMP results from the routers. NetFlow [14] is a tool developed for Cisco routers to track flows and is generally used with cflowd [2] to analyze network flows. CoralReef [3][12] provides current network traffic statistics as observed at a network location.

For all these tools configuration and control is usually restricted to network administrators, and only MRTG is commonly accessible by non-administrators. Generally, providing user access to tools like these is considered a security risk since they allow users access to other users' network traffic. Also, detailed information from SNMP regarding the state at each router is likely to be misinterpreted by people not familiar with the current configuration of the router and network. This can lead to a significant amount of incorrect incident reporting and unnecessary work on the part of the network administrators.

A similar work to SCNM is NIMI [17]. NIMI's goal is to create network measurement infrastructures from diversely administered hosts, providing clients with different types of access privileges. Security and scalability are NIMI's main focus.

There are also tools available to passively record detailed TCP packet traces. Among these, one of the most popular is *tcpdump* [9], which is usually used on application end-hosts to capture network traffic. The Bro [15] system is a passive network sensor that is used primarily to detect network-based intrusion attempts. Both Bro and CoralReef were specifically designed to operate on high-bandwidth streams (OC-48 in the case of CoralReef).

Another important aspect of network monitoring is data presentation tools. There are many such tools available; for understanding TCP streams, two useful ones are *tcptrace* [19] and *tcpanaly* [16]. Tcptrace is used to convert a tcpdump file into a graph that can be displayed by *xplot*. Tcpanaly can be programmed to interpret the network stream and diagnose various problems with the communication.

Other related work has been done by the IETF Realtime Traffic Flow Measurement system (RTFM) working group [1]. RTFM is designed to measure traffic flows for traffic passing a given point in a network, and could be used to collect and process the usage data so as to provide information and reports which are useful for network engineering and management purposes.

## 3.0 System Design

SCNM is designed to securely allow both network engineers and application developers access to monitoring data from inside the network. The infrastructure consists of both hardware and software components.

The SCNM hardware component is the SCNM monitoring host, which was illustrated in Figure 1. The monitoring host is installed at critical points in the network (e.g: next to key routers), where it will passively capture packet headers for application traffic. The current SCNM monitoring hosts can successfully capture packet headers on networks with speeds up to and including Gigabit Ethernet. For security reasons, the monitoring host is designed to be incapable of sending its own traffic on the network it is monitoring. Therefore, host administration and transmission of monitoring results is performed through a separate, typically lower bandwidth, network connection.

The software components of SCNM, shown in Figure 2, are the activation packet generator, data collection daemon, network analysis program (*SCNMPlot*), packet capture daemon (*pktd*), and capture configuration / data forwarding module. The first three of these components run on the application host, or "endpoint" host; the rest of the components run on the monitoring host. Below we describe how each of these components are used.

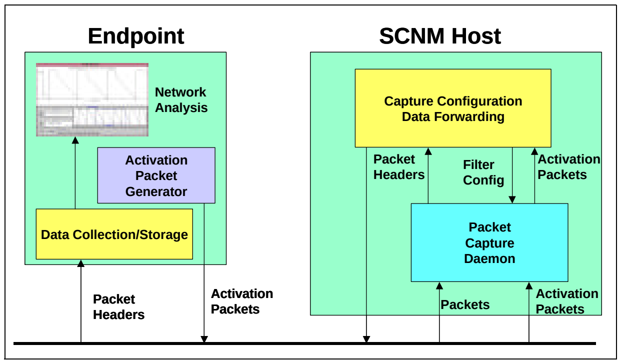

Figure 2: SCNM Software Components

A user wishing to activate monitoring of a particular data stream uses the activation packet generator to format and send an *activation packet*. The request is sent to the application destination endpoint, not the SCNM monitoring host. Thus the user does not need to explicitly know the locations and identities of the SCNM hosts on the path. All the SCNM monitoring hosts listen for these special UDP *activation packets* on a well-known port. Each SCNM host along the data path capture the activation packet as it travel past its interface.

The activation packet specifies the characteristics of the traffic to monitor including source, destination, and port(s) of the traffic. The format of a activation packet is illustrated below in Table 1. We use a thirty-two bit magic number field to permit quick recognition of valid SCNM packets, and to discard spurious packets on the activation port. The activation packet format version and the sequence number of the activation packet uniquely identify this request and how to interpret it. Traffic type provides the IP protocol number of the type of traffic to monitor (e.g. UDP or TCP). The rest of the packet contains parameters specifying the characteristics of the traffic to monitor such as the source and destination addresses and ports used by the traffic. For example, the destination address and port are specified using a parameter that is six bytes in length with the first four being address and the second two being the port.

| | | | |
|---|---|---|---|
| Magic Number | | | |
| Major Version | Minor Version | Sequence Number | Traffic Flags |
| Traffic Type | Number of Parameters | Parameter Type | Parameter Size |
| Parameters | | | |

Table 1: Activation packet format

Upon receipt of a UDP packet from the SCNM activation port, the SCNM software first verifies the magic number. If the magic number is correct, then the rest of the fields are checked. An activation packet is rejected if the specified source and destination of the traffic to be monitored does not match the source and destination in the IP header of the activation packet. If all fields are valid, the SCNM host configures itself to monitor that traffic by composing a filter string defining the traffic in the capture configuration software. The filter string is then set in the packet capture daemon.

The traffic-monitoring filter associated with a particular request automatically times out and the monitoring ceases after a predefined (and generally short) period of time unless another activation packet is received. The result is that stopping the monitoring requires no additional intervention on the part of the requester and no intervention on the part of the network operators. In addition, it is robust to host failure. To keep a filter active, the SCNM activation packet generator periodically resends the activation packet.

The resulting data is buffered and compressed on the SCNM monitor host, and then it is sent to either the source or destination of the monitored traffic (as specified by the activation packet).

Monitoring of a traffic flow is accomplished by capturing network- and transport-layer headers corresponding to a particular filter string in the SCNM monitor host. In order to support monitoring for several concurrent monitoring requests at the same SCNM host, we multiplex the network device. Network device multiplexing is complicated by configuration issues and operating system limitations. For example, Berkeley Packet Filter [11] (*BPF*)-based platforms limit the number of processes listening to the network device to one process per virtual *BPF* file.

We use the packet capture daemon, *pktd* [5], to provide multiple processes with shared access to a single virtual *BPF* device (*pktd* is based on *libpcap*[8], the standard packet capture library, which is also used by tcpdump, NIMI, and Bro). *Pktd* sets the device filter to a logical OR of all the requested filters, and the capture length (*snaplen*) to the largest requested capture length. When a packet matches the combined filter, it is captured by *pktd,* and delivered to the data forwarder.

## 4.0 Security Model

The SCNM host is designed to be installed, administered, and maintained by the network administrators with minimal effort. At the same time, an appropriate subset of its functionality is intended to be accessible to average users. Users can monitor particular data streams without requiring special access privileges or intervention of the network administrator.

The core of the security model revolves around the concept that a user is allowed to monitor only their own data. In order to be accepted by the SCNM host, the activation packet must be traveling between the source and destination of the traffic to be monitored. The SCNM host verifies this by comparing the request parameters with the source and destination in the IP header of the activation packet. Also, the SCNM host is only willing to send resulting data to the source or destination of the monitored traffic. Thus, although spoofing of the IP source and destination might result in an extra stream being monitored, the resulting monitoring data will not be sent to the spoofing host. Each SCNM host also maintains a local audit log of all monitoring requests.

In the current design, we assume that all users with access to an end host are allowed to monitor traffic to or from that host. If this turns out to be an issue, we can limit the ability of users to activate the monitor by using a privileged port for the activation packets. In this case, the end user would need to have root access to request monitoring of traffic to or from that host.

**Third-party monitoring requests**

Currently, requests are only accepted if they come from one of the end-hosts and the data can only be returned to one of the end hosts. This limitation simplifies the security model but makes it difficult to monitor traffic from hosts that are unable to receive the data. It also limits access to the data. In the next phase of SCNM, request packets will be enhanced to also allow inclusion of authorization credentials and a destination address. The network administrator of an SCNM host will be able to explicitly specify the set of allowed credentials and destination hosts. Each SCNM host will verify the credentials as they are received and initiate monitoring only if the request is authorized. In this case, both the client initiating the packet monitoring request and the data sink for the results may be different from the endpoints of the traffic to be monitored.

The administrator of each SCNM host may define a set of authorized users by defining a set of authorized keys that can sign requests coming from an arbitrary or specific location and request a test. To activate monitoring from a host that is not one of the endpoints requires a signed and authorized activation packet. This mechanism will allow activation of the monitor by network administrators and configuration of a dedicated monitoring archive for an SCNM host.

## 5.0 Implementation Issues

In this section, we describe some of the issues we encountered while implementing and deploying SCMN monitoring hosts. We show that standard PC hardware is capable of this type of passive monitoring.

### 5.1 Packet Capture Host Issues

Our currently deployed packet capture hosts are based on FreeBSD with a SuperMicro 370DL motherboard with a single P-III Xeon 933MHz CPU, PCI 64bit/66MHz IO bus, 133 MHz DIMMs, and two SysKonnect PCI 64 bit 1000BT network interface cards (NIC), one for capturing packets in each direction. Although these are relatively high-end systems, there were many performance related issues that had to be addressed. Most of these issues are relevant regardless of the power of the CPU. Modern high-end PC hardware requires significant tuning to enable capture and filtering of traffic at Gigabit per second speeds without dropping packets. This tuning included issues involving interrupt moderation, timestamping packets in the NIC, and reducing kernel memory copies, all discussed below. Using this hardware we have monitored TCP flows of up to 940 Mbits/sec for several minutes.

**Packet Capture Data Path**

When capturing high-bitrate packet streams, every packet takes the following path. Packets are captured by the network interface, which then copies the packets to system memory (mbufs) using DMA. Once the packet is in system memory, the CPU receives an interrupt to activation packet processing. The CPU then filters and trims the packets to leave only the IP and TCP headers. The result is then copied to one of two BPF kernel buffers. Data continues to be added to the buffer until it is full. When a BPF buffer becomes full, its contents are copied by BPF to the user space, handed to *pktd*, and then demultiplexed to the data forwarding component.

**Interrupt Moderation**

Capture of a network packet generates an interrupt from the NIC, so that the CPU can copy the packet from system buffer into BPF kernel buffer. Capturing packets on a 1000BT NIC generates one interrupt every 12 µs. The host system is not able to keep up with this interrupt rate. Some network cards, including the SysKonnect card, provide an interrupt moderation feature, also known as interrupt coalescence, which bundles several packets into a single interrupt. The idea is that the NIC, on receipt of a packet, does not automatically generate an interrupt to request a transfer of the data to memory. Instead, the interrupt is delayed for up to a given amount of time (the interrupt moderation period) in hopes of other packets arriving in the meantime and being served by the same interrupt.

Ideally, the interrupt moderation period is short enough to keep the NIC from running out of buffers and to avoid large delays in packet processing. The main drawback of interrupt coalescing is that the kernel is no longer able to assign accurate timestamps to the arriving packets. The problem is that packets reach the BPF kernel buffer a significant amount of time after they were copied to the system buffer. Fortunately, some network cards (including SysKonnect) have an onboard timestamp register which can provide information on the exact packet arrival time, and pass this timestamp to the system buffer descriptor. The unmodified FreeBSD BPF implementation obtains a timestamp from the system clock before processing and copying each packet into the BPF kernel buffer. We have modified the FreeBSD SysKonnect driver and BPF implementations to use the NIC timestamp, instead of the system clock timestamp.

The maximum interrupt interval depends on the average packet size and the space the kernel reserves for incoming data from the NIC, which is specified as a number of per-packet kernel receive buffers (N). The maximum interrupt interval can thus be calculated as:

```
time = N * average_packet_size /
 line_speed
```

The default number of kernel receive buffers for the SysKonnect driver is 256, and the default interrupt servicing frequency is 200 µs in the FreeBSD kernel. If the average packet size is 1000 bytes, 256 receive buffers will become occupied after 2 ms. If instead the packet size is l500-bytes, the interrupt interval can be as large as 3 ms. We are currently using a 1 ms interrupt moderation period. Increasing the number of receive buffers will allow a longer coalescence period, however, a large number of receive buffers requires large system memory resources.

**Memory Bandwidth**

Memory bandwidth is one of the most critical factors in data capture. After packets arrive at the SCNM host system, the packets must be copied from the NIC to the system memory. At this point the data is filtered, so only the packet headers which match a monitoring request filter are copied. The filtered headers are copied to the BPF buffer, then to user space, secondary storage, back to user space, and back to the system memory again before being copied to the NIC and sent to the endpoint. If memory is sufficient, the copy to secondary storage and back actually involves only one memory copy, therefore in practice the headers are copied 5 times.

For 1500 byte packets, the IP and TCP headers typically account for roughly 3% of the data. Assuming bidirectional traffic, each of the two NICs transfer data to the system bus at 125 MBytes/second. This means that the memory bandwidth required is:

$$2 \times 125\text{MBps} + [(2 \times 125\text{MBps} \times 3\%) \times 5\text{copies}] \cong 300 MBps$$

Capturing smaller packets requires even more bandwidth, as the ratio of header size to data size is much higher.

The first generation hardware used for the SCNM monitoring hosts were equipped with 133 MHz DIMMs. These DIMMs provide a system memory bandwidth of approximate 300 MBytes/second. This bandwidth satisfies monitoring of large packet traffic such as bulk data transfer, but it is far less than would be required to capture the headers of small packets. In practice, this hardware is capable of capturing all packets on a 1000BT network which is 80% utilized and has an average packet size of 800-bytes. Higher utilizations can be monitored if the average packet size is larger.

## 5.2 Software Design Issues

In order to perform high-bitrate packet capture, the SCNM software and kernel software such as BPF, required a great deal of optimization and tuning. In this section we discuss how we optimized the data path, including the use of buffered I/O and tuning of the BPF buffer size. We also discuss compression techniques we are using to reduce the data volume.

**Optimizing the Data Path**

All the code running at the SCNM host must be as efficient as possible to ensure the minimum amount of work is carried out on a per-packet basis.

BPF filtering is carried out after all the data has been copied from the NIC to the system buffers. Note that we are using the NIC in promiscuous mode, allowing the NIC to capture everything. The BPF kernel module compares each packet to the *pktd* filter, and any matching packet is trimmed and its IP and TCP headers are copied into the BPF kernel buffer.

When the BPF buffer becomes full, the kernel copies data from the BPF buffer to the capture daemon buffer (in user space). There are two BPF buffers, and the BPF module will start to fill the second BPF kernel buffer while the data is being copied from the first buffer. Therefore if the packet capture daemon cannot consume the data fast enough, the BPF module will drop packets when both BPF kernel buffers are full. Once the packet header is in user space, *pktd* delivers the headers to the data forwarder.

**BPF Buffer Size**

Another important aspect when trying to maximize the performance of packet capture is the size of the BPF kernel buffer. While this parameter is not exposed by *libpcap*, we have found it is an important factor in allowing high-bitrate packet capture. (Note: by high-bitrate we are referring to roughly 800 Mbps streams.) The current *libpcap* default value of 32 KB works well only when the amount of data captured per packet (the *snaplen*) is small (less than 100 bytes).

For larger snaplens, the BPF buffers have space for a smaller number of packets, which implies more context switches to move the data from the BPF buffers to *libpcap*. The consequence is that *libpcap* starts losing packets. For example, only half of the packets are captured when the snaplen is 300 bytes. The solution in this case was to increase the BPF buffer size. We found by experimentation that a BPF buffer size of about 140 KBytes produced much better results for the particular monitoring host we are using.

**Header Compression**

Even though we are only capturing headers, the amount of data collected can still be quite large. For example, a 10 second capture of a 300 Mbps TCP stream contains over 20 MB of header data. This is enough data to affect the measured link when it is sent back to the requestor. To reduce this load we have implemented compression of the headers.

In general, our compression implementation follows the approach used by CSLIP guidelines [7], but with some differences. First, CSLIP relies on the underlying link layer for conveying packet size information, whereas our compression must be fully self-contained. Second, we need to include timestamps as well as the packet headers. Third, CSLIP was designed to operate primarily in environments where space was at a greater premium than computation cycles, so squeezing every last bit of compression, at the cost of some non-aligned operations, makes sense for CSLIP. For *pktd*, we're more concerned with saving cycles than with squeezing every bit. Accordingly, for our compression we rely on byte operations instead of bit-wise ones.

We are also experimenting with selective compression, where some packet fields are compressed, possibly even in a lossy way and other fields are thrown out. The idea is to take advantage of the fact that the user may not need all the information present in the headers. For example, if we want to look for the source of lost packets, we probably do not need the TCP urgent pointer, nor any other TCP option. Others do not need to be replicated perfectly, and may be compressed in a lossy way. For example, possibly the only interesting information for checksums is whether or not it is correct. If the value of the IP checksum is wrong, the packet will be discarded by the kernel.

We found that our header compression works quite well: *pktd* achieves an average lossless compression ratio of about 6:1 for UDP headers and about 4:1 for TCP headers. The UDP headers, including timestamps, are represented with just 7 bytes/packet; the TCP headers, including timestamps and TCP options, in only 15 bytes/packet. The UDP traces were for synthetic, heavily-fragmented UDP packets with an average length of 1500 bytes, corresponding to a single connection. The TCP traces are from measured FTP traffic, with an average 4.16 bytes of options per packet. Compression is carried out with no noticeable performance loss. Using a 68-byte snaplen, we can support multiple concurrent *pktd* clients capturing the test stream with no packet losses (we've tested up to a dozen). For more information see [5].

## 6.0 DMZ access issues

Since the DMZ provides the site’s connection to the WAN, access to it is usually tightly controlled by the network administrators. A critical consideration in the design of SCNM was to create a security model that was acceptable to sites and would allow the monitors to be deployed on the DMZ.

To date, we have deployed SCNM monitoring hosts at the DMZs of Lawrence Berkeley National Laboratory (LBNL), Oak Ridge National Laboratory (ORNL), Stanford Linear Accelerator Center (SLAC), and National Energy Research Scientific Computing Center in Oakland, CA. The SCNM security model underwent an extensive security review at each site. In each case we met either personally or over the phone with the network administrators from the site. In this meeting we presented the SCNM security model and answered site questions. In all cases the existing SCNM security model was acceptable to the site once all their questions had been answered.

The issues encountered deploying SCNM hosts to large sites and convincing site security staff to install the SCNM host on their DMZ have varied from site to site. For example, one site took strict control over the machine including all user accounts, software upgrades, and responsibility for patches to the operating system. Several other sites requested that we administer the machine and take responsibility for upgrades and patches. Some of these sites wanted root or user access to the box to monitor activity or check access logs. In several of the installations the site provided the fiber taps for the DMZ. In a few we provided the taps. One of the other issues encountered was host installation and software upgrade timing. In several cases, we provided the SCNM host to the site in anticipation of final approvals. This sometimes meant that by the time the machine was installed its software was out of date. We generally resolved this by upgrading it once it was accessible via the network.

One of the sites expressed concern regarding the third party activation model and the assumption that everyone with access to a machine can monitor traffic to and from that machine. This site has, however, deployed the monitor and we will consult with them before deploying any third party activation capabilities to their site. We are also currently working with several additional sites to install more of these monitoring hosts.

## 7.0 Results

Preliminary sample results are shown below. For this test we used SCNM hosts installed at the LAN-WAN boundaries at LBNL and ORNL. In Figure 3, the line on the left is the packet trace from the SCNM monitoring host near the source (LBNL), and the line on the right from the SCNM host near the destination (ORNL). From the difference in slope of the two lines, one can clearly see that the path between the two monitoring hosts is fairly congested.

Figure 4 shows a zoomed-in view of this data. From this figure it is clear that the packets are much more unevenly spaced by the time they reach the ORNL DMZ.

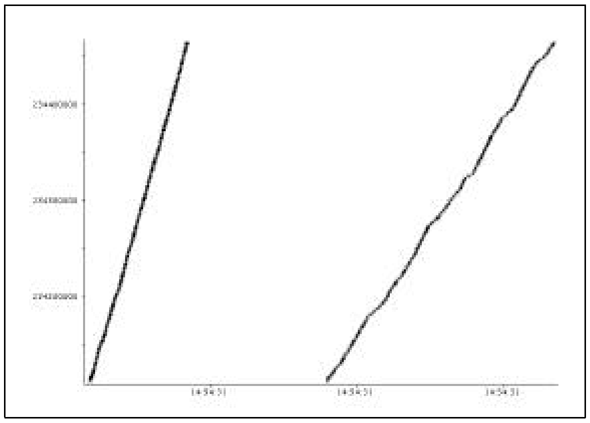
Figure 3: SCNMPlot output showing congested router

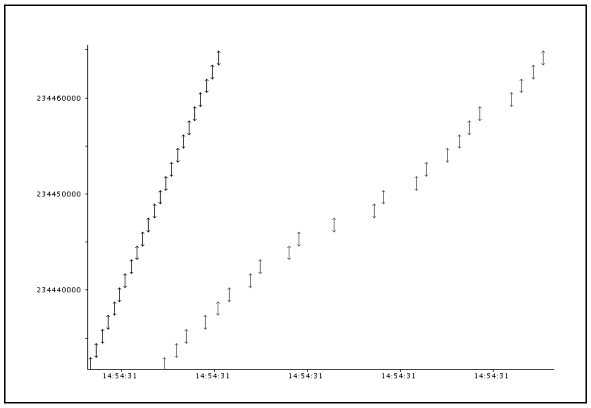

Figure 4: Zoomed view

Later in the trace, Figure 5 shows that a large burst of packets are lost after they leave site 1 (LBNL), and before they arrived at site 2 (ORNL).

These plots were generated by using tcptrace to create xplot files from the tcpdump-formatted files collected by SCNM. Our modified version of the jPlot [10] (a Java version of *xplot*), called *SCNMPlot*, supports comparison of multiple input files. *SCNMPlot* also allows one to shift packet traces in time for easy visual analysis.

This type of data analysis and visualization can be invaluable in helping users and network administrators to find the location and nature of network problems.

## 8.0 Future Work

As we install SCNM monitoring hosts at more sites across the network, we expect that packet header trace analysis will reveal other interesting network behavior. We expect to be able to detect phenomena such as packet reordering, packet compression (clustering together of packets) [18], and so on.

We do not explicitly support NAT (Network Address Translation) [4]. This is not an issue as our measurement hosts are located close to the ISP router. However a future direction could be to support monitoring of traffic whose identification changes depending on the measurement point.

10 Gigabit Ethernet is now readily available, and 10 Gigabit PCI-X NICs are just becoming available [6]. Capturing traffic at this rate will be much more challenging, but we believe this could be possible. For example, to capture ten 200 Mbit/second flows (35% link utilization of a 10 Gigabit Ethernet) requires 908 MB/s memory bandwidth. The best current PC hardware (early 2003) is approaching this memory bandwidth. If the link utilization is higher or the packet size is smaller, the capture system will require more memory bandwidth. One solution that could help is to do header compression earlier in the filtering pipeline; at the BPF filtering phase.

## 9.0 Conclusions

By allowing end users to monitor their own traffic, the Self-Configuring Network Monitoring system provides critical functionality needed to diagnose network problems. The architecture allows users to trigger passive monitoring hosts in the interior of the network without needing intervention by network administrators. Early results show that SCNM monitoring data can be used to help identify congested or misconfigured network segments. We are currently working on improvements to the efficiency of the software to allow capture of higher rate streams and deployment of the monitor to additional sites.

## 10.0 Acknowledgments

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-51846.

## 11.0 References

[1] N. Brownlee "RTFM: Applicability Statement", IETF RFC2721

[2] Cflowd: http://www.caida.org/tools/measurement/cflowd

[3] CoralReef:http://www.caida.org/tools/measurement/coralreef/

[4] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)", RFC 1631, May 1994

[5] J.M. Gonzalez and V. Paxson, "*pktd*: A Packet Capture and Injection Daemon", Proceeding of the Passive and Active Monitoring Workshop, April 2003.

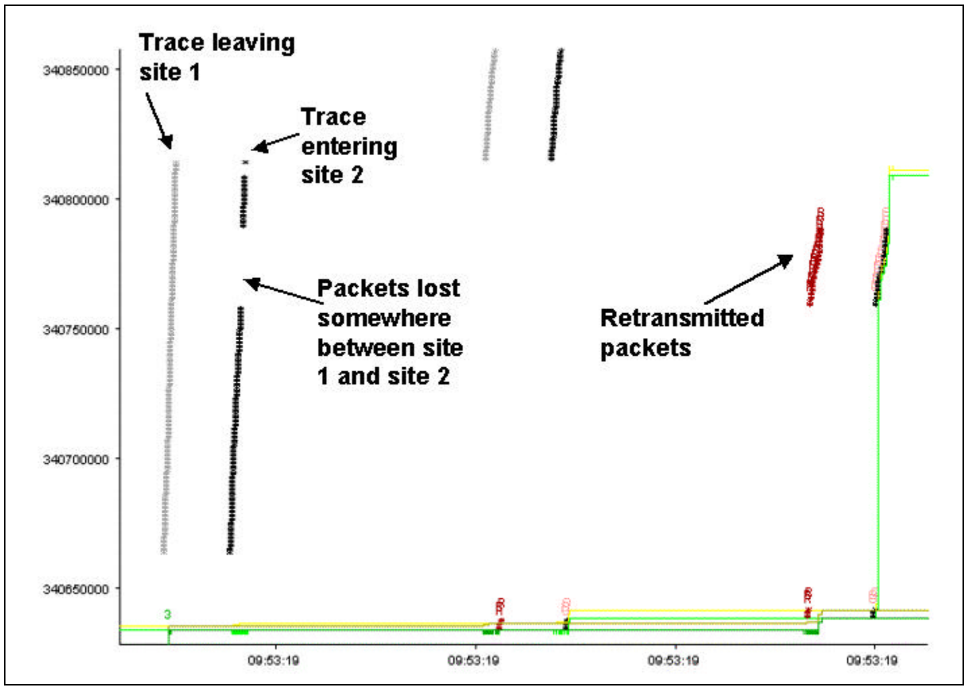

Figure 5: SCNMPlot output showing large burst of lost packets

[6] Intel PRO/10GbE LR Server Adapter http://www.intel.com/network/connectivity/products/10gigabit/index.htm

[7] V. Jacobson. “Compressing TCP/IP headers for low-speed serial links”, 1990.

[8] V. Jacobson and C. Leres and S. McCanne, “pcap: Packet Capture library. UNIX man page,” 1993 (http://www.tcp-dump.org)

[9] V. Jacobson, C. Leres, and S. McCanne, “tcpdump: dump traffic on a network. UNIX man page,” 1993. (http://www.tcp-dump.org)

[10] jPlot: http://www.tcptrace.org/jPlot/

[11] S. McCanne and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture,” Proc. Winter USENIX Technical Conference, Jan. 1993.

[12] D. Moore, et. al., “CoralReef software suite as a tool for system and network administrators”, Usenix Lisa Conference, 2001

[13] MRTG: The Multi Router Traffic Grapher, http://people.ee.ethz.ch/~oetiker/webtools/mrtg/

[14] NetFlow whitepaper: http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm

[15] V. Paxson, “Bro.: A System for Detecting Network Intruders in Real-Time,” Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998.

[16] V. Paxson, “Automated Packet Trace Analysis of TCP Implementations,” ACM SIGCOMM'97, September 1997, Cannes, France.

[17] V. Paxson, A. Adams, and M. Mathis, “Experiences with NIMI”, Proc. Passive & Active Measurement: PAM-2000.

[18] V. Paxson, “Measurements and Analysis of End-to-End Internet Dynamics”, Ph.D. dissertation, 1997.

[19] Tcptrace: http://www.tcptrace.org/